



AN APPLICATION BASED EFFICIENT THREAD LEVEL PARALLELISM SCHEME ON HETEROGENEOUS MULTICORE EMBEDDED SYSTEM FOR REAL TIME IMAGE PROCESSING

K. INDRAGANDHI* AND P.K. JAWAHAR†

Abstract. The recent advent of the embedded devices is equipped with multicore processor as it significantly improves the system performance. In order to utilize all the core in multicore processor in an efficient manner, application programs need to be parallelized. An efficient thread level parallelism (ETLP) scheme is proposed in this paper and uses computationally intensive edge detection algorithm for evaluation. Edge detection is the important process in various real time applications namely vehicle detection in traffic control, medical image processing etc. The main objective of ETLP scheme is to reduce the execution time and increase the CPU core utilization. The performance of ETLP scheme is evaluated with basic edge detection scheme (BEDS) for different image size. The experimental results reveal that the proposed ETLP scheme achieves efficiency of 49% and 72% for the image size 300×256 and 1024×1024 respectively. Furthermore an ETLP scheme reducing 66% execution time for image size 1024×1024 when compared with BEDS.

Key words: Heterogeneous, Multicore Processor, Image processing, Edge Detection, CPU utilization

AMS subject classifications. 68W10, 94A08

1. Introduction. Multicore processor plays a drastic effect in the digital world. In earlier days, single core processors were in market in which performance was achieved by increasing the frequency and transistors count per chip. The dynamic power dissipation (Pd) of CMOS chip is given in equation (1.1):

$$Pd = CV^2f \quad (1.1)$$

where C is the capacitance, V is the supply voltage, f is the frequency. From the equation 1, it was observed that the frequency is directly proportional to the power dissipation. When we increase the frequency of single core processor to get good performance, the chip will dissipate more power which in turn increases the heat [1]. In 1990, Processor performance was increased by 60% per year but from 2000 to 2004, it was reduced to 40% per year with only 20% increase in performance. In a single core processor, multi tasking was based on time slice which will degrade the performance if more number of multiple tasks wants to be executed parallelly [2]. Compared to many high frequency single core processor, multicore processor chip will dissipate less heat since it has simpler CPU core. Multicore processor means many cores are fabricated in a single socket in which each core operating in same or different frequency. Performance is achieved in multicore chip by allocating the task parallelly on all cores. Parallel processing is easily achieved in multicore chip compare to multiple single core chips with low cost. The main feature of multicore chip is that tremendous increase in performance by increasing the number of cores instead of increasing the frequency [3]. To improve the multicore CPU performance, three factors namely parallelism granularity, incorrect programming model and language compilers to be tuned [10].

Nowadays, most of the embedded applications are parallel processing application. To improve the performance, an application must be multithreaded. But multithreading an application is a time consuming work. Programmers concentrate mostly on the efficiency of an application. Since most of the popular programming languages support the sequential code, the programmers prefer to write an application in sequential manner. Using the automatic parallelization tool eg. Par4All, Cetus, TRACO etc., the sequential code is converted into

*B.S.Abdur Rahman Crescent Institute of Science and Technology, Chennai,Tamil Nadu, India. (indra@crescent.education – corresponding Author)

†B.S.Abdur Rahman Crescent Institute of Science and Technology, Chennai,Tamil Nadu, India. (jawahar@crescent.education)

parallel application. If the code is too complex, the compiler may wrongly parallelize the code which may increase the execution time [7]. For a last decade, Computer vision field has a drastic growth but the need for real time image processing is not fulfilled by conventional sequential computing. These leads to parallel processing high performance computing. The image processing and signal processing programmers can benefit dramatically from the advances in multicore processor, by modifying single threaded code to multithreaded code [5]. Xin Gao [18] proposed a post processing schemes for vehicle detection in wide area aerial imagery, the proposed scheme gives better performance for this kind applications.

In this paper, we propose an Efficient Thread Level Parallelism (ETLP) scheme to utilize the CPU cores efficiently in an embedded device. ETLP scheme is applied based on the application. ETLP scheme provides very good efficiency for multithreaded application.

2. Related Work. Hahn Kim et al. [4] provides a survey of software technologies for image and signal processing on multicore architectures. The programmer need not to aware of the complexity of multicore architectures, instead of that, they can focus on application algorithms. Parallel programming languages, high level languages, general purpose programming on GPUs and middleware libraries were described and their efficiencies are compared.

Marungo et al. [8] presented a extended version of OpenMP parallel programming model for a multi-processor system on chip embedded system. The task were scheduled among the accelerators and host using OpenCL and OpenMP run time environment. The algorithm was evaluated in STMicroelectronics STHORM development board using standard benchmark applications. The results shows 30x speed up using OpenMP compared to OpenCL.

S. Mittal and J. Vetter [9] surveyed the heterogeneous computing using CPU and GPU processor. The performance of heterogeneous computing technologies are reviewed at algorithm, programming, compiler, runtime and application levels.

T.Singh et al. [11] developed the tools to measure performance and CPU core utilization for multicore architecture. The quick sort with different set of data elements are executed serially and parallelly in dual and quad core CPU. The compared results are represented as graph using those tools and exploiting more parallelism provide better CPU core utilization.

A.Goyal et al. [12] presented a comparative study of edge detection algorithm using sobel, prewitt and canny filter on satellite images using OpenACC, OpenMP, MPI and hybrid OpenMP/MPI model in multicore architecture. The comparative results showed that the edge detection algorithm using OpenACC achieved greatest speedup over other models.

S. Bernabé et al. [13] developed a new parallel un mixing chain for hyper spectral images sensing in multicore processors. The optimization were achieved using OpenMP and Intel Math Kernel Library for real time image processing on CPU-GPU architecture. The results reveal better performance compared to GPU architecture.

A. Monteria et al. [14] developed a small, low cost and lightweight convolution neural network(CNN) – embedded device using Raspberry Pi 3 to find faults in structural health monitoring system. The case study were evaluated experimentally using piezoelectric patches glued over an aluminum plate and results reveal 100% effective hit rate.

G. Tagliavini et al. [15] proposed a fine grained parallelism using a parallel programming model OpenMP for embedded programmable many core accelerators. The run time environment (RTE) design were implemented in Kalray MPPA 256 and the results showed an average speed up of 12x for real benchmarks compared with traditional scheduling of tasks.

A.Nadjaran et al. [16] presented CLOUDS-Pi platform, a low cost software defined cloud data center for research purpose using raspberry pi embedded computers integrated with OpenvSwitch to develop a network of open flow switches.

W. Kwedlo et al. [19] proposed four approaches namely Drake's, Elkan's, Annulus, and Yinyang algorithms to minimize the unnecessary distance calculation in K-means clustering algorithm. In this paper a hybrid MPI/OpenMP programming models are used to parallelize these algorithms. The results of proposed method showed efficiency in computing time compared to Lloyd's algorithm.

M.K. Pekturk et al. [20] developed a spectral angle mapper and matched filter algorithm using OpenMP

programming model for online remote sensing applications. The experimental result reveals that 3-4 times & 16-19 times improvement for offline and online data processing.

3. Proposed Method. The main idea of proposed Efficient Thread Level Parallelism (ETLP) scheme is to exploit the full processing power of multicore processor by using parallel programming model for multithreaded application. The proposed system implemented in raspberry pi 3 embedded device for image processing application to increase the performance and CPU core utilization. In our method edge detection application is chosen for evaluation. The ETLP scheme is achieved using parallel programming model OpenMP. The OpenMP has a runtime library routines, a set of pre-processor directives and environment variables. The ETLP scheme is implemented based on the multithreaded application. The programmer want to identify which section of application source code can be multithreaded and single threaded, based on that, we want to use the appropriate OpenMP pragmas, runtime library routines, etc., The proposed scheme is evaluated for edge detection process using sobel, prewitt and scharr operator for different image size and compared with Basic Edge Detection Scheme (BEDS). BEDS represents edge detection execution in sequential approach. The proposed ETLP scheme shows the better performance, efficiency and CPU core utilization compared to BEDS.

4. Parallel programming model – OpenMP. A single core processor execute only one task at a time in a sequence manner and it may take hours or days to execute complex application. A multicore processor can execute many task parallel which help to solve complex application with less time and cost compared to single core processor. OpenMP is an powerful application programming interface, those API’s will support more functionalities needed for parallel programming .Manual parallel execution is achieved by using the parallel programming model. The programming model will act as a bridge between software and hardware. Some of the parallel programming models are Pthread, MPI, OpenMP, CUDA etc. In the proposed method, OpenMP parallel programming model is used to improvise the performance of embedded device. OpenMP is designed for shared memory based multicore architecture.

OpenMP provide explicit full control to parallelize the code for the programmers. Using threads, OpenMP parallelize the code. The thread is a small unit of processing, scheduled among the cores. OpenMP follow fork- join model. From Semr Aydin et al. [17], a fork is the master thread begin executing serial region while encountered the parallel region, it creates group of independent threads and all the threads are executed parallel.

Once the parallel execution over, the results are synchronized and joined as a master thread to execute serial region of the code as shown in Fig.. 4.1. The OpenMP API are available in C and FORTRAN language. In the proposed method C language based API is used. The general C structure of OpenMP is shown in Fig. 4.2.

OpenMP directives are selected based on various parts of source code. Before implementing the OpenMP directives, analyse the code carefully, because improper selection of OpenMP directives may degrade the performance and execution time may get increased.

5. Image Processing – Edge detection. An image is a 2-D function $f(x, y)$, where f is the intensity and (x, y) are coordinates of pixel. If an image is finite and discrete, then it is known as digital image. Digital image processing helps to exploit the digital image in the computer system. Image segmentation is the part of

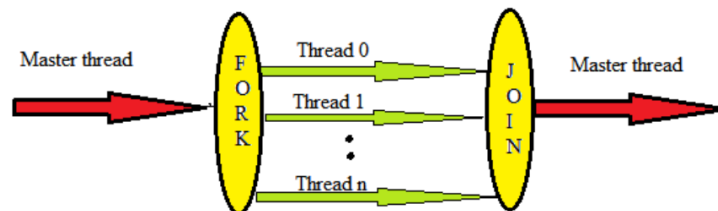


FIG. 4.1. Fork – join model of openMP

```

#include <omp.h>
main () {
serial execution code
parallel region begin
Fork a threads as group.
#pragma omp parallel
{
All the threads execute parallel
Run-time Library calls
Other OpenMP directives
....
....
All threads are joined as a master thread
}
Resume serial execution of code
}

```

FIG. 4.2. General C structure of openMP

TABLE 5.1
Edge detection operators

Direction	Sobel Operator	Prewitt Operator	Scharr Operator
Horizontal	$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}$
Vertical	$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$	$\begin{bmatrix} +1 & + & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$

image processing. Image segmentation guides to extract the useful information from the image. It is typically used to detect the edges, curves and boundaries of object in an image. Edge detection is one of the essential processes of image segmentation. Edges are substantial changes of intensity in an image. The boundary between two different segments in an image is called as edges. The four basic steps of edge detection are smoothing, enhancement, detection and localization. Edge detection is done by two methods namely first order derivative and second order derivative. Edge points in an image can be detected by finding the minima and maxima of first derivative method or detecting zero crossing of second derivative method. Spatial kernel operator is used to detect the edges in horizontal and vertical direction of an image.

In the proposed ETLP scheme, Edge detection is carried out by gradient method of first order derivative using sobel, prewitt and scharr operator. To generate the gradient of an image, convolute the input image with 3X3 kernel matrix of particular operator. Table 5.1 represents the kernel matrix of sobel, prewitt, scharr operator for horizontal and vertical edge detection.

$$\text{Horizontal derivative approximation } (G_x) = 3 \times 3 \text{ kernel} * f(x, y) \quad (5.1)$$

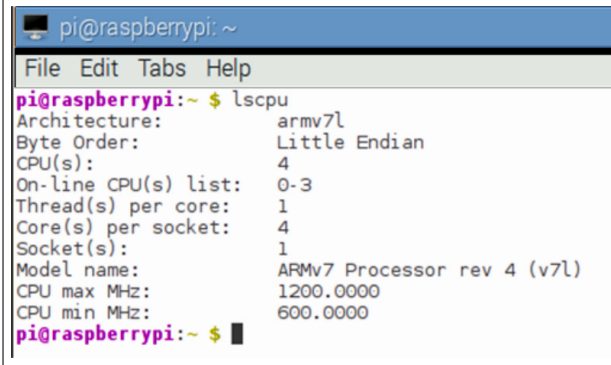
$$\text{Vertical derivative approximation } (G_y) = 3 \times 3 \text{ kernel} * f(x, y) \quad (5.2)$$

$$\text{Gradient magnitude } (G) = \sqrt{G_x^2 + G_y^2} \quad (5.3)$$

where $f(x, y)$ represent input image and $*$ stand for 1-D convolution operation

6. Hardware description. Raspberry Pi 3 is a single board heterogeneous embedded device, which includes BCM2837 Broadcom processor, 1.2GHz quad core 64-bit ARM Cortex A53 processor, maximum stressed power consumption of 1W and 1GB RAM [14]. Since it is a quad core processor, four tasks can

run parallelly. By principle, the performance is four times greater than the original speed, but in real time it is difficult to make use of full potential of multicore processor. To utilize the full processing power of quad core processor, the proposed ETLP scheme uses the parallel programming model OpenMP. The proposed method helps to improve the execution speed by using software optimization in a raspberry pi 3 embedded devices. Raspbian is the primary operating system for the Raspberry Pi family boards. To implement parallel programming model, CPU architecture should be well known . Fig. 6.1 shows Raspberry Pi 3 CPU architecture.



```

pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ lscpu
Architecture:      armv7l
Byte Order:        Little Endian
CPU(s):            4
On-line CPU(s) list: 0-3
Thread(s) per core: 1
Core(s) per socket: 4
Socket(s):         1
Model name:        ARMv7 Processor rev 4 (v7l)
CPU max MHz:       1200.0000
CPU min MHz:       600.0000
pi@raspberrypi:~ $ █

```

FIG. 6.1. CPU architecture details of Raspberry Pi 3

7. ETLP scheme. The edge detection algorithm is written in C language and compiled in GCC compiler of raspbian operating system. Before implementing ETLP, analyze the application source code and identify the private variable, shared variable and critical section. Loop level parallelism is achieved only if there is no loop dependence. The main focus is to increase the performance without affecting the expected result. So we should use the OpenMP cautiously. The pseudo code for BEDS is given in Fig. 7.1. In the code Step 4 and 5 has nested four for loops and it is a time consuming tasks. The critical section, shared and private variables are mentioned in Fig. 7.1. Race condition is one of the major issues in thread level parallelism. Race condition will occur when multiple threads try to access the same shared resource, these leads to deadlock. Critical section is a piece of code, which access the shared variables. Only one thread is allowed to enter the critical section for execution at a time. Threads are get synchronized at this point. The kernel operator is Sobel, Prewitt or Scharr.

Fig. 7.2 is the pseudo code of ETLP scheme. Based on the code analysis of Fig. 7.1, the OpenMP pragma, run time libraries are used in the code. Collapse clause instructs the compiler to collapse the first two for loops. The collapsed for loops should not have any instruction between them. Raspberry Pi3 is having a quad core processor, so threads are equally distributed among the cores using OpenMP dynamic scheduling mechanism. In a dynamic schedule, based on the workload the iterations are distributed among the threads at runtime. In the proposed method, dynamic scheduling with chunk size 50 is used, representing 50 iterations are distributed among the threads dynamically based on the availability of core during run time.

The proposed ETLP scheme is evaluated with sobel, prewitt and scharr operator for different image size. Four different image size of 300×246 , 512×512 , 720×526 and 1024×1024 are selected for evaluation. Fig. 7.3 shows the original image and edge detected image using BEDS and ETLP scheme. Fig 7.3.a is the original image of size 720×526 , Fig 7.3.b and c are edge detected image using BEDS and ETLP scheme respectively. From the figure it is observed that the edge detected output is same for BEDS and ETLP scheme.

The edge detection application is executed three times and an average time was calculated for evaluation of proposed method. Different sizes of images are used to highlight performance of ETLP scheme. Table 7.1 represents the time taken for different image size using sobel, prewitt and scharr operator.

Fig. 7.4 Shows the execution time comparison for BEDS and ETLP scheme of different image size. Figs. 7.4.a, b and c are comparison of execution time using Sobel, Prewitt and Scharr operator respectively. From the Fig. 7.4.a it is observed that for image size 300×246 the execution time is 0.33 s for BEDS and 0.166 s for ETLP scheme, nearly 50% execution time is reduced by the proposed method. Similarly for image size

```

Algorithm 1: Basic Edge Detection Scheme (BEDS)
1. Input image <- IP [a] [b]
2. Width<- w, Height<- h, Maximum<-M & Minimum<-m
3. 3X3 Kernel for horizontal direction <-KX [c][d] & vertical direction <-KY[c][d]
4. Calculating the gradient value(G) //G,Gx,Gy are private variable and M & m are shared
for a<- 1 to w{ //Nested four for loop
for b<- 1 to h{
for c<- -1 to <=1{
for d<- -1 to <=1{
Gx +<- KX(d+1)(c + 1) * IP(a + c)(b + d)
Gy +<- KY(d+1)(c + 1) * IP(a + c)(b + d)}}
G<-square root of ((Gx * Gx) + (Gy *Gy));
if G is less than m then , m <- G //critical section
if G is greater than M) then, M<- G;}}
5. Output image<-OM [a] [b] //G,Gx,Gy are private variable and M & m are shared
for a<- 1 to w{ //Nested four for loop
for b<- 1 to h{
for c<- -1 to <=1{
for d<- -1 to <=1{
Gx + <-KX(d+1)(c + 1) * IP(a + c)(b + d)
Gy + <-KY(d+1)(c + 1) * IP(a + c)(b + d)}}
G<- square root of ((Gx * Gx) + (Gy *Gy))
S<- maximum brightness * (G-m)/(M-m)
OM[a-1][b-1]<-S}}

```

FIG. 7.1. Code structure of BEDS

```

Algorithm 2: ETLP scheme
1. Include OpenMP header file
2. omp_set_num_threads(4) <- To utilize the quad core processor
3.Start time<-omp_get_wtime( )
4.Input image <- IP[a] [b]
5. Width<- w, Height<- h, Maximum<-M & Minimum<-m
6. 3X3 Kernel for horizontal direction<-KX [c][d] & vertical direction <-KY[c][d]
7. Calculating the gradient value
#pragma omp parallel for private(G,Gx,Gy,c,d) shared(M,m) collapse(2)
schedule (dynamic,50)
for a<- 1 to w{ //Nested four for loop
for b<- 1 to h{
for c<- -1 to <=1{
for d<- -1 to <=1{
Gx +<- KX(d+1)(c + 1) * IP(a + c)(b + d)
Gy +<- KY(d+1)(c + 1) * IP(a + c)(b + d)}}
G<-square root of ((Gx * Gx) + (Gy *Gy));
#pragma omp critical
{if G is less than m ,then m <-G
if G is greater than M ,then M<- G;}}

8.Output image<-OM [a] [b]
#pragma omp parallel for private(G,Gx,Gy,c,d) shared(OM) collapse(2)
schedule (dynamic,50)
for a<- 1 to w{
for b<- 1 to h{
for c<- -1 to <=1{
for d<- -1 to <=1{
Gx + <-KX(d+1)(c + 1) * IP(a + c)(b + d)
Gy + <-KY(d+1)(c + 1) * IP(a + c)(b + d)}}
G<- square root of ((Gx * Gx) + (Gy *Gy))
S<- maximum brightness * (G-m)/(M-m) OM[a-1][b-1]<-S}}
9.End time<-omp_get_wtime();
10.Execution time <- End time- Start time

```

FIG. 7.2. Code structure for ETLP scheme

1024 × 1024 the execution time is 4.679 s for BEDS and 1.712 s for ETLP scheme, nearly 63% execution time is reduced. Thus the proposed method gives better performance for large amount of data.

The performance of proposed method is evaluated based on speed up, performance improvement and efficiency using execution time for sequential and parallel execution [6]. The expression for speedup, performance improvement and efficiency are given in equations (7.1), (7.2) and (7.3).

$$Speedup = \frac{Execution\ of\ Sequential\ method\ (s)}{Execution\ of\ Parallel\ method\ (s)} \quad (7.1)$$

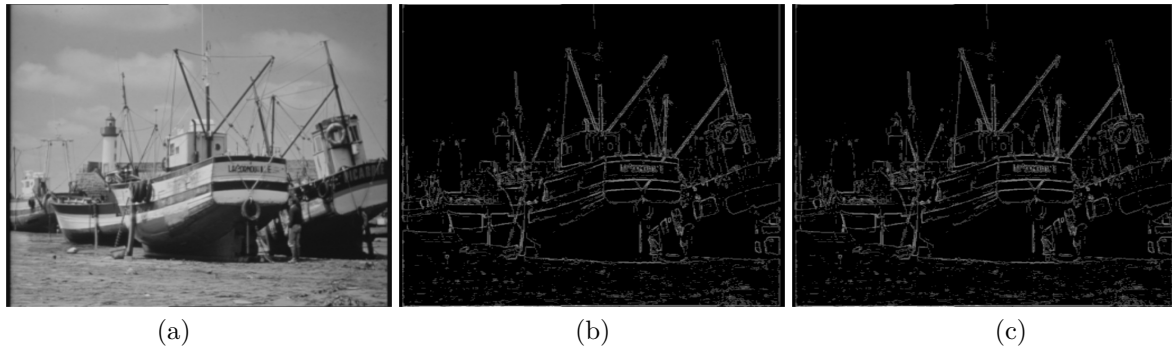


FIG. 7.3. Comparison of edge detection output using BEDS and ETLP scheme: (a) Original image 720×526 ; (b) Edge detected with BEDS; (c). Edge detected with ETLP scheme

TABLE 7.1

Execution time in seconds for different image size using different operator for BEDS and ETLP scheme

Image Size	Iteration	Sobel		Prewitt		Scharr	
		BEDS	ETLP	BEDS	ETLP	BEDS	ETLP
1024X1024	1	4.785	1.688	4.799	1.711	4.805	1.679
	2	4.781	1.744	4.833	1.684	4.865	1.685
	3	4.47	1.704	4.798	1.726	4.784	1.656
	Average	4.679	1.712	4.81	1.707	4.818	1.673
720X526	1	1.889	0.745	1.912	0.702	1.889	0.711
	2	1.893	0.716	1.917	0.685	1.954	0.705
	3	1.891	0.709	1.93	0.717	1.924	0.706
	Average	1.891	0.723	1.92	0.701	1.922	0.707
512X512	1	1.233	0.475	1.023	0.456	1.217	0.474
	2	1.232	0.492	1.219	0.46	1.21	0.45
	3	1.236	0.477	1.205	0.475	1.221	0.46
	Average	1.234	0.481	1.149	0.464	1.216	0.461
300X246	1	0.346	0.162	0.313	0.156	0.342	0.199
	2	0.376	0.172	0.315	0.169	0.343	0.151
	3	0.344	0.163	0.312	0.17	0.345	0.172
	Average	0.355	0.166	0.313	0.165	0.343	0.174

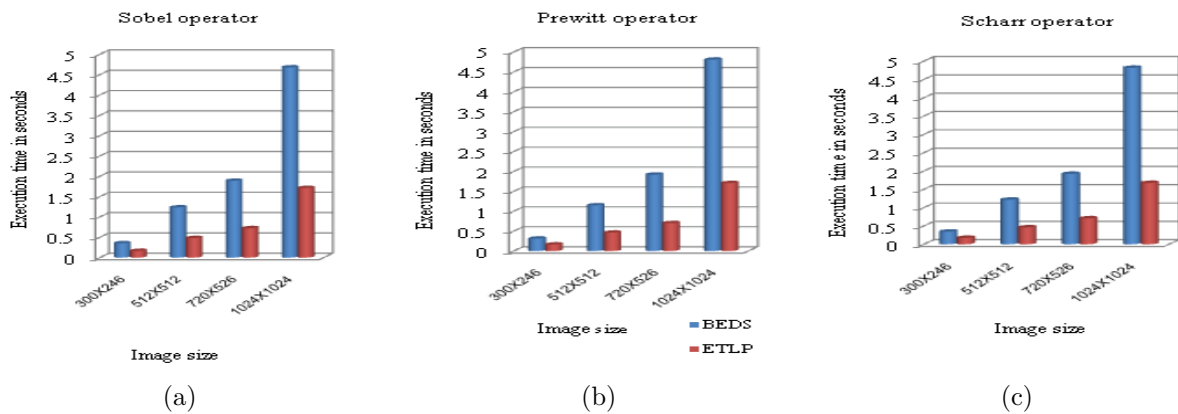


FIG. 7.4. Comparison of execution time for BEDS and ETLP scheme using different operators and different image size: (a) Sobel operator; (b) Prewitt operator; (c) Scharr operator

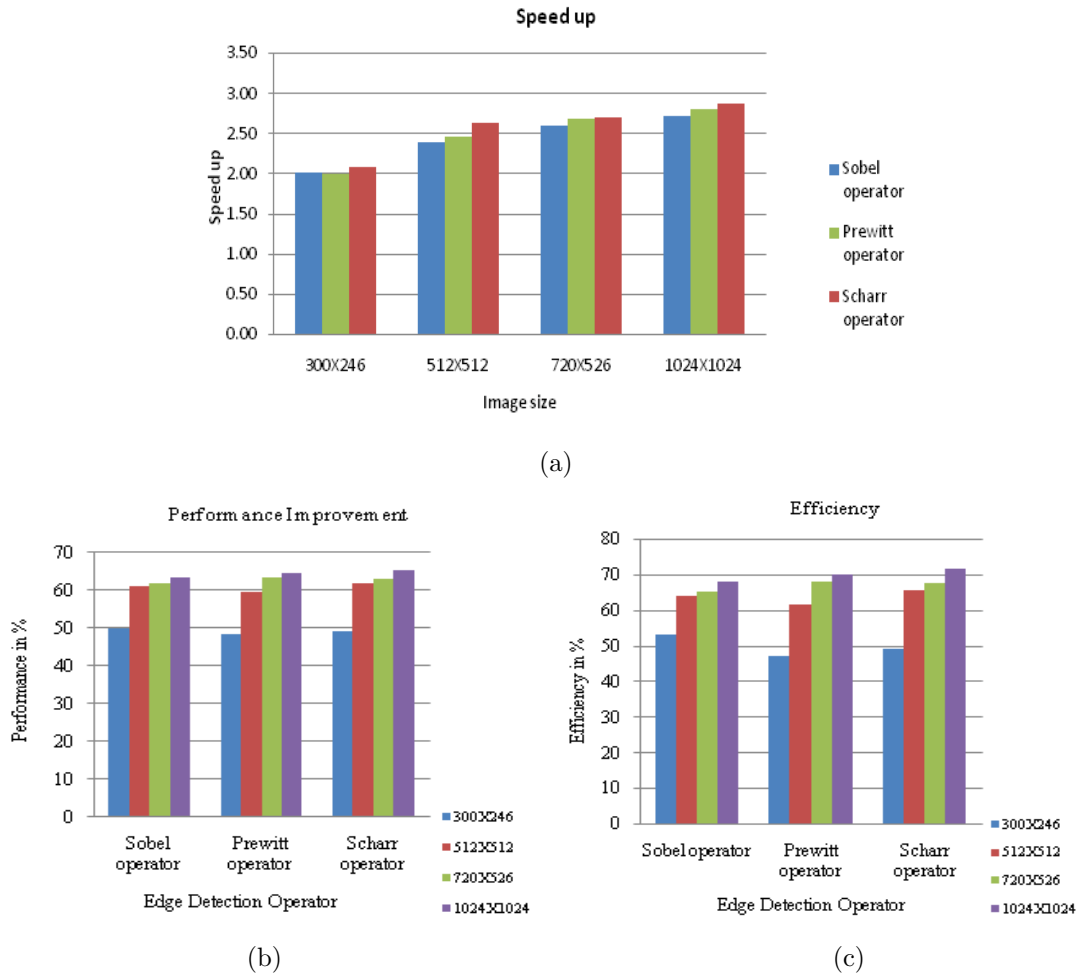


FIG. 7.5. Performance evaluation of ETLP scheme: (a) Speedup (b) Performance Improvement (c) Efficiency

$$\text{Performance improvement} = \frac{\text{Execution of Seq method (s)} + \text{Execution of } l^{\text{el}} \text{ method(s)}}{\text{Execution of } l^{\text{el}} \text{ method (s)}} \quad (7.2)$$

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Number of Processors}} \quad (7.3)$$

In our proposed method, Raspberry Pi 3 embedded board is used which consists of quad core processor. In equation 7.3, number of processor is four. Fig. 7.5 Shows the performance evaluation of proposed method. Figs. 7.5.a, b and c show the speed up, performance improvement and efficiency respectively for different image size and different operator. From Fig. 7.5.c, it is observed that the efficiency of proposed method using sobel, prewitt and scharr operator for the image size 1024×1024 are 68.33%, 70.45% and 72% respectively. Table 7.2 is the performance evaluation of image size 1024×1024 .

The main objective of proposed method is not only increase the efficiency also increase the individual CPU core utilization. Fig. 7.6 shows the comparison of CPU core utilization for various image size using different operator. Figs. 7.6.a, b, c and d shows the CPU core utilization for image size 300×246 , 512×512 , 720×256 and 1024×1024 . Consider Fig 7.6.b, sobel operator based BEDS method, the CPU0, CPU1, CPU2 & CPU3 utilization is 39%, 2%, 2% & 0% respectively. Even though the processor having four CPUs, only CPU0

TABLE 7.2
Sample performance evolution of ETLP scheme for image size 1024X1024

Performance Evaluation for 1024X1024 image			
Edge detection operator	Speed up	Performance Improvement	Efficiency
Sobel	2.73	63.411	68.33
Prewitt	2.82	64.51	70.45
Scharr	2.88	65.27	72

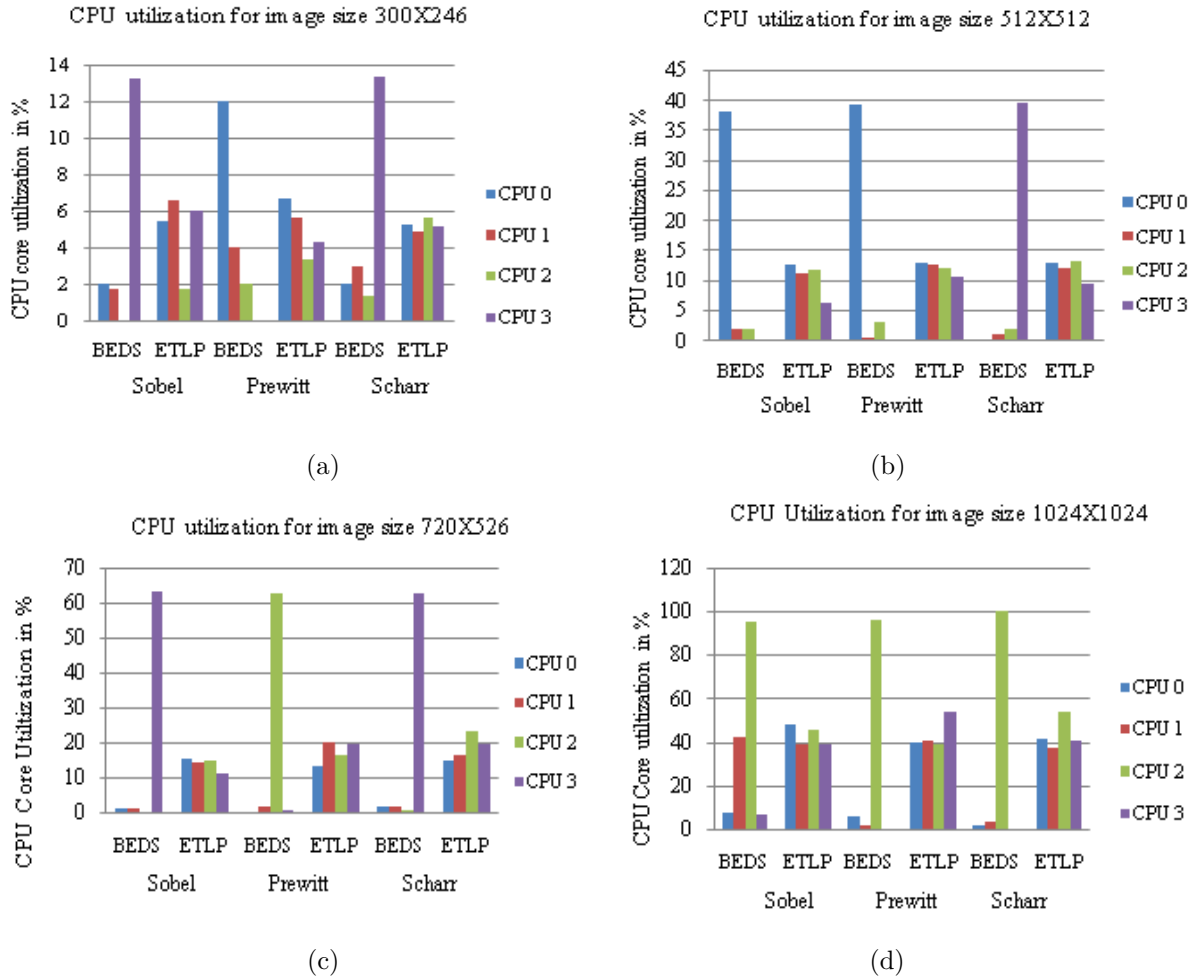


FIG. 7.6. Comparison of CPU core utilization of BEDS and ETLP scheme for different image size and operator: (a) CPU utilization for image size 300×246 ; (b) CPU utilization for image size 512×512 ; (c) CPU utilization for image size 720×256 ; (d) CPU utilization for image size 1024×1024

is utilized more compare to others and CPU3 is in idle state. In sobel operator based ETLP scheme, the CPU0,CPU1,CPU2 &CPU3 utilization is 13%,11%,12% & 6% respectively. Thus the proposed method utilized the CPU core efficiently by distributing the thread among the CPU core.

8. Conclusion and Future work. The main ultimate goal of many embedded device is, how to increase the performance of the application without degrading the expected outcome. The life time of multicore processor may reduce if the threads are not evenly distributed among the CPU core. The proposed ETLP scheme shows better performance compare to traditional method of execution. Multimedia applications based embedded

device get more advantages since the most of the operations are multithreaded. The proposed ETLP scheme is implemented in quad core and evaluated for different image size. From the experimental results, it is concluded that the proposed method shows the gradual increase in performance and CPU cores are evenly utilized for small to large image size. The performance of proposed ETLP scheme can be further extended for larger parallel data and many CPU core architecture.

REFERENCES

- [1] D. GEER. (2005) ‘Chip Makers Turn to Multicore Processors’, *Computer*, vol. 38, pp. 11-13.
- [2] P.F. GORDER. (2007), ‘Multicore Processors for Science and Engineering’, *Computing in Science & Engineering*, IEEE Computer Society, vol.9, Issue:2,ISSN: 1521-9615, pp.3-7.
- [3] G. BLAKE, R. G. DRESLINSKI, AND T. MUDGE. (2009) ‘A survey of multicore processors’, *IEEE Signal Process. Mag.*, vol. 26, no. 6, pp. 26–37.
- [4] HAHN KIM, AND R. BOND. (2009), ‘Multicore software technologies’ ,*Signal Processing Magazine*, IEEE ,pp. 80-89.
- [5] G. SLABAUGH, R. BOYES AND X. YANG. (2010), ‘Multicore Image Processing with OpenMP’, *IEEE Signal Processing Magazine*, vol. 27, no. 2, pp. 134-138.
- [6] HARON N., AMIR R., AZIZ I.A., JUNG L.T., SHUKRI S.R. (2010) , ‘Parallelization of Edge Detection Algorithm using MPI on Beowulf Cluster’, *Innovations in Computing Sciences and Software Engineering*, pp. 477-482.
- [7] PER LARSEN, RAZYLADELSK, JACOB LIDMAN, SALLY A. MCKEE, SVEN KARLSSON AND AYALZAKS. (2012), ‘Parallelizing More Loops with Compiler Guided Refactoring’,*41st International Conference on Parallel Processing*, pp-410-419.
- [8] A. MARONGIU, A. CAPOTONDI, G. TAGLIAVINI AND L. BENINI. (2015) , ‘Simplifying Many-Core-Based Heterogeneous SoC Programming With Offload Directives’ , *IEEE Transactions on Industrial Informatics*, vol. 11, no. 4, pp. 957-967.
- [9] S.MITTAL AND J.VETTER. (2015), ‘A Survey of CPU-GPU Heterogeneous Computing Techniques’, *ACM Computing Surveys*, vol.47, no.4, pp.1-35.
- [10] TRONG-TUAN VU, BILELDERBEL. (2016) , ‘Parallel Branch-and-Bound in multi- core multi- CPU multi-GPU heterogeneous environments’, *Elsevier journal, Future Generation Computer Systems*, pp. 95–109.
- [11] T. SINGH, D. K. SRIVASTAVA AND A. AGGARWAL. (2017) , ‘A novel approach for CPU utilization on a multicore paradigm using parallel quicksort’ , *3rd International Conference on Computational Intelligence & Communication Technology (CICT)*, Ghaziabad, pp. 1-6.
- [12] A. GOYAL, Z. LI AND H. KIMM. (2017), ‘Comparative Study on Edge Detection Algorithms Using OpenACC and OpenMPI on Multicore Systems’ , *IEEE 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoc)*, Seoul, pp. 67-74.
- [13] S. BERNABÉ, L. I. JIMÉNEZ, C. GARCÍA, J. PLAZA AND A. PLAZA. (2018), ‘Multicore Real-Time Implementation of a Full Hyper spectral Unmixing Chain’, *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 744-748.
- [14] A. MONTEIRO, M. DE OLIVEIRA, R. DE OLIVEIRA AND T. DA SILVA. (2018), ‘Embedded application of convolutional neural networks on Raspberry Pi for SHM’, *Electronics Letters*, vol. 54, no. 11, pp. 680-682.
- [15] G. TAGLIAVINI, D. CESARINI AND A. MARONGIU. (2018) , ‘Unleashing Fine- Grained Parallelism on Embedded Many-Core Accelerators with LightweightOpenMP Tasking’, *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 9, pp. 2150-2163.
- [16] A. NADJARANTOOSI, J. SON AND R. BUYYA. (2018), ‘CLOUDS-Pi: A Low-Cost Raspberry-Pi based Micro Data Center for Software-Defined Cloud Computing’ *IEEE Cloud Computing*, vol. 5, no. 5, pp. 81-91.
- [17] SEMRA AYDIN, REFIKSAMET, OMER FARUK BAY. (2018) , ‘Real-time parallel image processing applications on multicore CPUs with OpenMP and GPGPU with CUDA’ , *The Journal of Supercomputing*, vol. 74, no. 6, pp. 2255-2275.
- [18] XIN GAO (2018), ‘Vehicle detection in wide-area aerial imagery: cross- association of detection schemes with post-processing’s’, *International Journal of Image Mining*, 2018 Vol.3 No.2, pp.106 - 116.
- [19] W. KWEDLO AND P. J. CZOCHANSKI, ”A Hybrid MPI/OpenMP Parallelization of K - Means Algorithms Accelerated Using the Triangle Inequality,” in *IEEE Access*, vol. 7, pp. 42280-42297, 2019
- [20] M. K. PEKTURK, Y. OZUZUN AND O. OZSANCAKTAR, ”Implementation of SAM and MF Algorithms with Multi-Core Programming,” 2019 9th International Conference on Recent Advances in Space Technologies (RAST), Istanbul, Turkey, 2019, pp. 619-625.

Edited by: Swaminathan JN

Received: Sep 27, 2019

Accepted: Dec 9, 2019